OAuth 2.0 - Guia Completo com Exemplos

OAuth 2.0 (Open Authorization)

OAuth 2.0 (Open Authorization) e um protocolo de autorizacao que permite a um utilizador conceder acesso limitado a recursos protegidos a uma aplicacao, sem ter de partilhar as suas credenciais. Representa uma alternativa segura e flexivel aos metodos tradicionais de autenticacao.

Principais beneficios do OAuth 2.0

Seguranca:
-> O utilizador nao precisa de partilhar as suas credenciais (como nome de utilizador e palavra-passe) com a aplicacao.
-> O processo de autorizacao e realizado atraves de um servidor de autorizacao, responsavel por verificar a identidade do utilizador e conceder o acesso ao aplicativo.

Flexibilidade:
-> Suporta varios tipos de aplicacoes: web, moveis e desktop.
-> Oferece diferentes fluxos de autorizacao.

Facilidade de uso:
-> O utilizador apenas precisa de clicar num botao para conceder acesso a aplicacao, sem digitar manualmente as suas credenciais.

Principais conceitos do OAuth 2.0
- Utilizador: O proprietario dos recursos a que se pretende aceder.
- Aplicacao (Cliente): A aplicacao que solicita acesso aos recursos do utilizador.
- Servidor de Autorizacao: Responsavel por autenticar o utilizador e conceder os tokens de acesso a aplicacao.
- Servidor de Recursos: Hospeda os recursos protegidos (como ficheiros, dados pessoais, etc.).

Fluxos de autorizacao do OAuth 2.0
- Codigo de Autorizacao: O mais seguro, utilizado por aplicacoes web que podem manter confidencialidade.

- Implicito: Um fluxo mais simples, adequado para aplicacoes do lado do cliente (menos seguro, hoje em desuso).
- Concessao de Recursos (Resource Owner Password Credentials): Utilizado quando a aplicacao precisa diretamente das credenciais do utilizador (uso desaconselhado atualmente).
- Credenciais do Cliente (Client Credentials): Utilizado quando nao ha intervencao do utilizador, ideal para comunicacao entre servidores.

Exemplo de uso do OAuth 2.0

Imagine que um utilizador quer guardar um ficheiro no Google Drive atraves de uma aplicacao web. A aplicacao redireciona o utilizador para o servidor de autorizacao do Google, onde este se autentica e concede permissao a aplicacao. Apos a autorizacao, a aplicacao recebe um token de acesso, que pode ser usado para comunicar com o servidor de recursos (Google Drive) e carregar o ficheiro.

Conclusao

O OAuth 2.0 e um protocolo essencial para garantir seguranca, praticidade e flexibilidade no acesso a recursos protegidos em aplicacoes web, moveis e desktop. Ele protege as credenciais dos utilizadores e permite que aplicacoes interajam com servicos externos de forma controlada e segura.

Grant Types do OAuth 2.0

Os grant types (tipos de concessao) no OAuth 2.0 sao os diferentes fluxos de autorizacao que definem como o cliente (aplicacao) obtem um token de acesso. Cada tipo e usado para diferentes cenarios e niveis de seguranca.

1. Authorization Code Grant

Uso: Aplicacoes web com servidor backend (confidenciais)

Funcionamento:

- O utilizador e redirecionado para o servidor de autorizacao.
- Apos login e consentimento, e devolvido um codigo de autorizacao.
- A aplicacao troca esse codigo por um token de acesso no backend.

[OK] Mais seguro, recomendado para aplicacoes com servidor.

OAuth 2.0 - Guia Completo com Exemplos

2. Implicit Grant (obsoleto / desaconselhado)

Uso: Aplicacoes frontend (sem backend), como SPAs (Single Page Applications)

Funcionamento:

- O token de acesso e retornado diretamente no redirecionamento.

- Nao ha codigo de autorizacao.

[!] Menos seguro (o token fica visivel na URL); ja nao e recomendado.

3. Resource Owner Password Credentials Grant (ROPC)

Uso: Aplicacoes de confianca total entre utilizador e cliente (ex.: apps proprias)

Funcionamento:

- O utilizador fornece nome de utilizador e senha diretamente a aplicacao.

- A aplicacao envia essas credenciais ao servidor de autorizacao para obter um token.

[!] Risco elevado desaconselhado em ambientes com terceiros.

4. Client Credentials Grant

Uso: Comunicacao entre servidores (sem utilizador humano)

Funcionamento:

- A aplicacao usa o seu proprio client ID e secret para obter um token.

- Nao ha interacao do utilizador.

[OK] Ideal para APIs, servicos internos ou aplicacoes backend.

5. Refresh Token Grant

Uso: Renovar tokens expirados sem exigir novo login

Funcionamento:

- Com um refresh token, a aplicacao pode obter um novo token de acesso.

[OK] Garante sessao continua sem incomodar o utilizador.

Resumo em Tabela:

| Grant Type | Interacao do Utilizador | Seguranca | Recomendado para |

OAuth 2.0 - Guia Completo com Exemplos

|----------|----------|----------|----------|

| Authorization Code | Sim | Alta | Web apps com backend seguro |

| Implicit (obsoleto) | Sim | Baixa | SPAs (mas ja substituido por PKCE) |

| Password Credentials (ROPC) | Sim (envio direto) | Baixa | Apps proprias e de confianca total |

| Client Credentials | Nao | Alta | Comunicacao entre servicos/servidores |

| Refresh Token | Nao (apos login inicial)| Alta | Sessoes longas e renovacao automatica |

Exemplos dos Fluxos do OAuth 2.0 com cURL e Python

1. Authorization Code Grant

Pre-requisitos:

- client_id, client_secret, redirect_uri

- Codigo de autorizacao (obtido apos redirecionamento do utilizador)

Trocar codigo por token:

cURL:

```
curl -X POST https://authorization-server.com/token \
  -d "grant_type=authorization_code" \
  -d "code=AUTH_CODE" \
  -d "redirect_uri=https://yourapp.com/callback" \
  -d "client_id=YOUR_CLIENT_ID" \
  -d "client_secret=YOUR_CLIENT_SECRET"
```

Python:

```
import requests

data = {
```

```
    "grant_type": "authorization_code",
    "code": "AUTH_CODE",
    "redirect_uri": "https://yourapp.com/callback",
    "client_id": "YOUR_CLIENT_ID",
    "client_secret": "YOUR_CLIENT_SECRET"
}

response = requests.post("https://authorization-server.com/token", data=data)
print(response.json())
```

---

2. Client Credentials Grant

Usado para apps sem utilizador, como servicos backend.

cURL:

```
curl -X POST https://authorization-server.com/token \
  -d "grant_type=client_credentials" \
  -d "client_id=YOUR_CLIENT_ID" \
  -d "client_secret=YOUR_CLIENT_SECRET"
```

Python:

```
data = {
    "grant_type": "client_credentials",
    "client_id": "YOUR_CLIENT_ID",
    "client_secret": "YOUR_CLIENT_SECRET"
}

response = requests.post("https://authorization-server.com/token", data=data)
print(response.json())
```

---

3. Resource Owner Password Credentials (ROPC)

Usado quando o utilizador fornece usuario e senha diretamente.

cURL:

```
curl -X POST https://authorization-server.com/token \
  -d "grant_type=password" \
  -d "username=USER" \
  -d "password=PASS" \
  -d "client_id=YOUR_CLIENT_ID" \
  -d "client_secret=YOUR_CLIENT_SECRET"
```

Python:

```
data = {
    "grant_type": "password",
    "username": "USER",
    "password": "PASS",
    "client_id": "YOUR_CLIENT_ID",
    "client_secret": "YOUR_CLIENT_SECRET"
}

response = requests.post("https://authorization-server.com/token", data=data)
print(response.json())
```

---

4. Refresh Token Grant

OAuth 2.0 - Guia Completo com Exemplos

Obter novo token sem novo login.

cURL:

```
curl -X POST https://authorization-server.com/token \
  -d "grant_type=refresh_token" \
  -d "refresh_token=YOUR_REFRESH_TOKEN" \
  -d "client_id=YOUR_CLIENT_ID" \
  -d "client_secret=YOUR_CLIENT_SECRET"
```

Python:

```
data = {
    "grant_type": "refresh_token",
    "refresh_token": "YOUR_REFRESH_TOKEN",
    "client_id": "YOUR_CLIENT_ID",
    "client_secret": "YOUR_CLIENT_SECRET"
}

response = requests.post("https://authorization-server.com/token", data=data)
print(response.json())
```